Data Version Control (DVC) in Machine Learning Projects

Nhóm MLOps

Ngày 18 tháng 10 năm 2025

Nội dung về được chia thành 5 phần chính:

- **Revision: Data Versioning và ETL pipeline**
- **Overview: Giới thiệu vòng đời của 1 dự án Học Máy**
- **Feature Store và Feast là gì ?**
- **Case Study**
- **Final Revision: ôn lại mọi khái niệm trong bài cùng team Time Series**

Phần 1: Từ ETL, Processing, Data Versoning đến Feature Store

Hành trình của dữ liệu bắt đầu từ quy trình **ETL (Extract, Transform, Load)**. Dây là bước nền tảng giúp thu thập, chuẩn hóa dữ liệu thô từ nhiều nguồn đa dạng (như các CSDL, Excel, Ảnh, etc..), sau đó làm sạch, biến đổi và lưu trữ chúng vào một trung tâm tập trung như Data Warehouse.

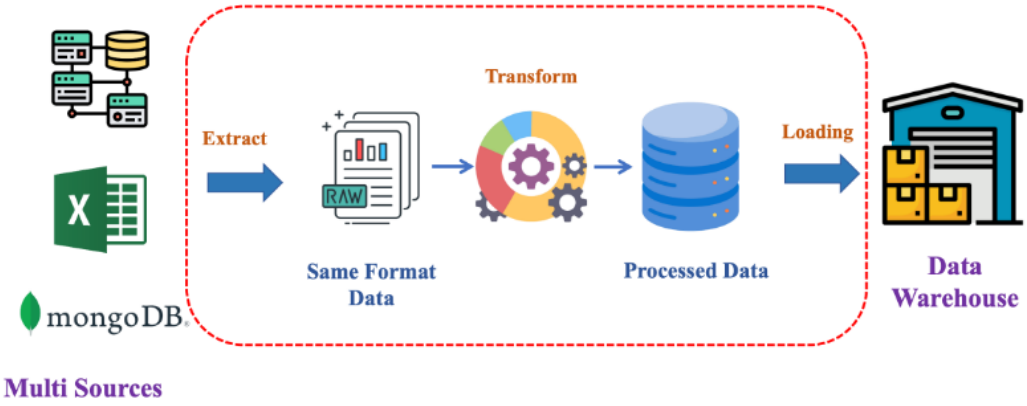


Figure 1: ETL pipeline

Giai đoạn tiếp theo là **Processing (Xử lý)**. Chúng ta không chỉ xử lý dữ liệu "tĩnh" (static) mà còn cả dữ liệu "động" (streaming data) như đơn hàng Shopee hay tin nhắn. Các công cụ như Apache Spark cho phép chúng ta xử lý và *tạo ra các đặc trưng (features)* từ dữ liệu lịch sử (ví dụ: "số tiền chi tiêu trung bình của khách hàng X trong 30 ngày qua"). Dây là các đặc trưng dùng để **huấn luyện (training)** mô hình.

Chính tại đây, thách thức lớn đầu tiên xuất hiện. Khi mô hình được huấn luyện bằng các đặc trưng offline được tải về để huấn luyện này (ví dụ: "số tiền trung bình 30 ngày" được tính 1 lần/ngày bằng Spark), làm thế nào để khi mô hình được **dự đoán (inference)** trong thời gian thực, nó có thể nhận được *chính xác* các đặc trưng đó (ví dụ: "số tiền trung bình 30 ngày" được tính ngay tại thời điểm 3:05:10 PM) ?

Hai quy trình này—một cho training (xử lý theo lô, offline) và một cho inference (xử lý thời gian thực, online)—thường được xây dựng độc lập. Điều này tất yếu dẫn đến hiện tượng **Training-Serving Skew**: dữ liệu mà mô hình "thấy" lúc huấn luyện khác với dữ liệu nó "thấy" lúc dự đoán.

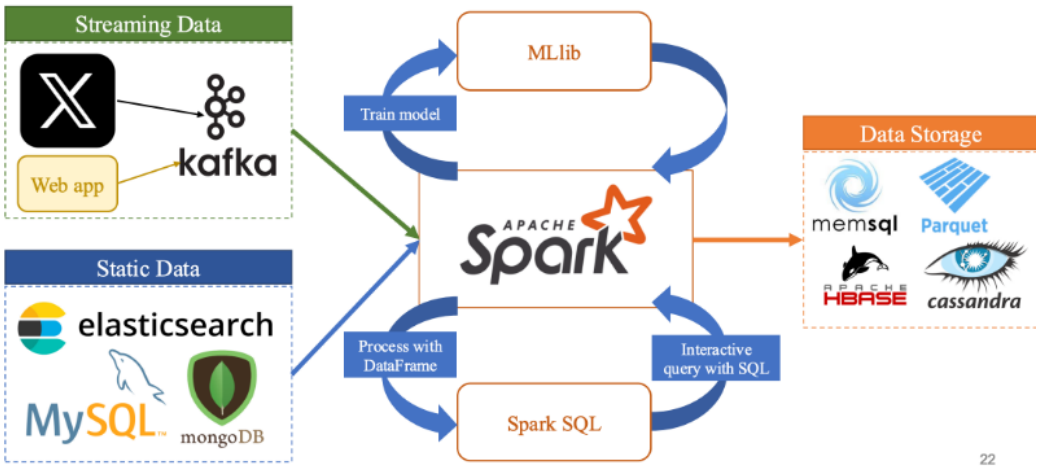


Figure 2: Process pipeline - Nơi bắt đầu xảy ra sự phân tách giữa pipeline offline (training) và online (serving).

Khi các mô hình trở nên phức tạp, chúng ta cần đến **Data Versioning**. Công cụ này giúp quản lý các "tạo tác" (artifacts) của quy trình *huấn luyện*, như phiên bản mô hình, tệp tham số, và các bộ dữ liệu đã được tạo ra. Nó đảm bảo khả năng tái lặp lại các thử nghiệm, tương tự như Git quản lý mã nguồn.

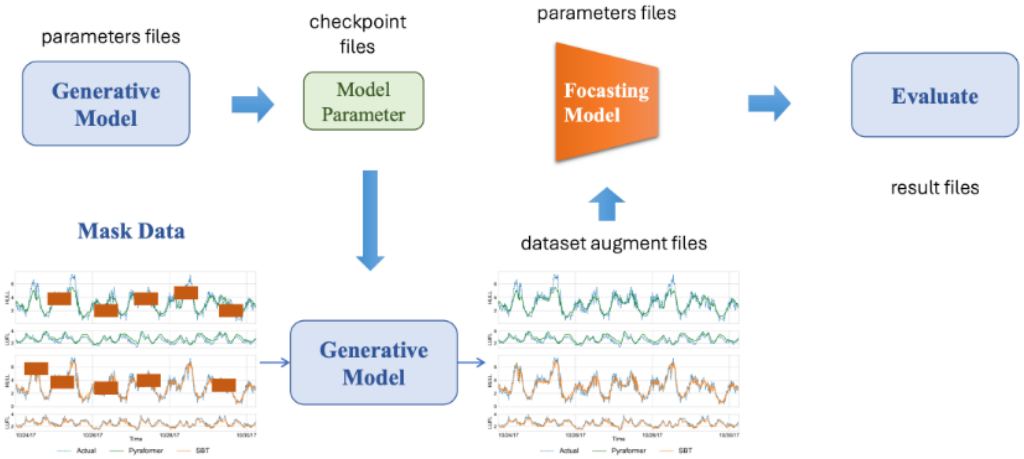


Figure 3: Data versioning Pipeline

Cuối cùng, chúng ta có một bức tranh rõ ràng:

1. **ETL** cung cấp dữ liệu thô.

2. **Processing** (Spark) tạo ra các đặc trưng offline cho *training*.

3. **Data Versioning** (DVC) quản lý các phiên bản mô hình được tạo ra từ *training*.

Tuy nhiên, vẫn còn một "lỗ hổng" chí mạng: Làm thế nào để cung cấp các đặc trưng cho *inference* (dự đoán) một cách nhất quán với các đặc trưng đã dùng để *training*? Đây chính là bài toán mà **Feature Store** ra đời để giải quyết. Nó là cầu nối duy nhất, đảm bảo tính nhất quán của đặc trưng cho cả hai môi trường.

Phần 2: ML/AI Project Lifecycle - Giới thiệu vòng đời của 1 dự án Học Máy

Để dễ hình dung vòng đời của 1 dự án học máy gồm 4 giai đoạn, qua một ví dụ tuần tự: **Xây dựng một hệ thống phát hiện gian lạn thẻ tín dụng theo thời gian thực**:

- **Data Engineering:** giải quyết vấn đề: *"Chúng ta có dữ liệu gì ? có những kiểu, đặc điểm gì ? dùng nó để huấn luyện như thế nào ?"*.
  → Kỹ sư dữ liệu sẽ thu thập lịch sử giao dịch (dữ liệu tĩnh) từ Data Warehouse rồi làm sạch và biến đổi chúng để tạo ra các "đặc trưng" (features) như "số tiền chi tiêu trung bình của người dùng này trong 30 ngày qua" hay "số lần chuyển tiền trong 1 ngày $>$ 20". Kết quả là một bộ dữ liệu sạch sẽ gồm những đặc trưng được kết hợp đại diện cho các yếu tố trong thực tế, sẵn sàng để huấn luyện.

- **Modeling:** giải quyết vấn đề: *"Làm sao để tìm ra quy luật của các gian lận, những yêu tố, đặc trưng (feature) nổi bật của gian lận đó ?"*.
  → Sau đó 1 nhóm gồm các Data Scientist sẽ nhận dữ liệu và huấn luyện các mô hình học máy hoặc học sâu, học các quy luật ngầm (patterns) đằng sau dữ liệu để phân biệt giữa 1 giao dịch bình thường và 1 giao dịch đáng ngờ (ví dụ: "số tiền lớn bất thường" + "vị trí lạ" → +"số lần chuyển tiền trong 1 ngày $>$ 20" -> 95% là gian lận).

- **Deployment:** giải quyết vấn đề: *"Làm sao để ứng dụng (appication) dùng được mô hình này NGAY LẬP TỨC?"*.
  → Khi đã có mô hình (ví dụ: file `model.pkl`), công việc triển khai không chỉ đơn giản là tải file này lên máy chủ. Thách thức lớn nhất là làm thế nào để **cung cấp dữ liệu đầu vào (features) cho mô hình** với tốc độ cực nhanh và phải *nhất quán* với dữ liệu đã huấn luyện. Đây là lúc hiện tượng **Training-Serving Skew** thể hiện rõ ràng nhất qua các tình huống cụ thể:

  - **Tình huống 1: Sai lệch về Logic (Logic Skew).**
    - *Khi Training (Offline):* Nhà khoa học dữ liệu dùng Python/Pandas để tính đặc trưng "số lượng giao dịch trong 1 giờ qua". Khi tính, họ có thể dùng hàm `.fillna(0)` để điền số 0 vào những khoảng thời gian không có giao dịch.
    - *Khi Serving (Online):* Kỹ sư phần mềm, để tối ưu tốc độ, có thể viết lại logic này bằng Java. Nhưng thay vì điền số 0, hệ thống online của họ lại *bỏ qua* (ignore) các khoảng thời gian đó khi tính toán.
    - *Kết quả:* Cùng một tên đặc trưng nhưng lại có hai giá trị khác nhau. Mô hình được huấn luyện với giá trị `0` nay lại nhận một giá trị khác, dẫn đến dự đoán sai.
  - **Tình huống 2: Sai lệch về Thời gian (Temporal Skew).**
    - *Khi Training (Offline):* Đặc trưng "số tiền chi tiêu trung bình 30 ngày" được tính theo lô (batch) và cập nhật vào lúc nửa đêm mỗi ngày.

    - *Khi Serving (Online):* Một khách hàng thực hiện giao dịch lúc 3 giờ chiều. Hệ thống online lấy đặc trưng "số tiền trung bình 30 ngày" được cập nhật lúc nửa đêm (dữ liệu đã cũ 15 tiếng).
    - *Kết quả:* Mô hình đang dự đoán dựa trên dữ liệu "cũ", trong khi lúc huấn luyện nó luôn được cung cấp dữ liệu "mới nhất" (tại thời điểm tính toán offline).
  - **Tình huống 3: Sai lệch về Giá trị Mặc định (Default Value Skew).**
    - *Khi Training (Offline):* Với một khách hàng mới chưa có giao dịch, script huấn luyện gán giá trị mặc định là `-1` cho đặc trưng "số tiền giao dịch cuối cùng".
    - *Khi Serving (Online):* API phục vụ, khi không tìm thấy giao dịch cho khách hàng mới, lại trả về giá trị `null` hoặc `0`.
    - *Kết quả:* Mô hình nhận được đầu vào mà nó chưa từng thấy trong quá trình huấn luyện, dẫn đến hành vi không đoán trước được.

  Do đó, công việc của Kỹ sư AI trong giai đoạn này là phải tích hợp mô hình, tối ưu phần cứng (CPU, GPU, CUDA) và **quan trọng nhất** là đảm bảo pipeline dữ liệu đầu vào cho inference phải tuyệt đối nhất quán với pipeline dữ liệu đã dùng cho training.

- **Business Analysis & Monitoring:** giải quyết vấn đề: *"Mô hình có đang hoạt động đúng và mang lại giá trị không?"*. Cuối cùng, team phân tích kinh doanh sẽ giám sát để đánh giá hiệu quả của liệu mô hình có đáp ứng sát yêu cầu của doanh nghiệp hay không qua các thử nghiệm rồi báo cáo lại với team Data Engineer để đưa ra nhận định và cải tiến.

  Bốn giai đoạn này tạo thành một vòng lặp liên tục, và **AI Infrastructure (Cơ sở hạ tầng AI)** chính là nền móng (máy chủ, cơ sở dữ liệu, các công cụ) giúp toàn bộ vòng lặp này hoạt động một cách nhanh chóng, bảo mật và có thể mở rộng.

Data Engineering

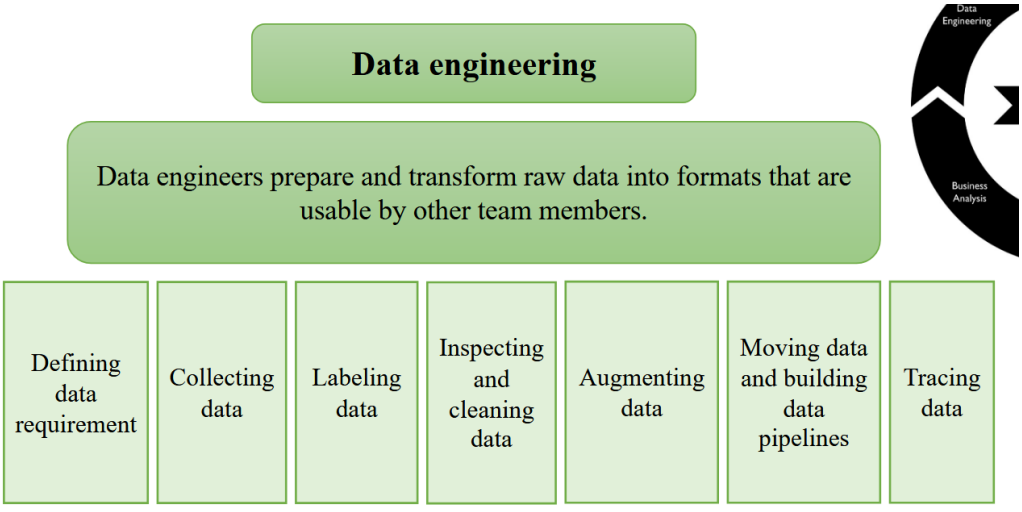


Figure 4: Data Engineer Pipeline

Trong ví dụ phát hiện gian lận, điều này có nghĩa là: **Defining data requirement** (định nghĩa dữ liệu giao dịch gồm những gì), **Collecting data** (thu thập dữ liệu từ hệ thống ngân hàng), **Inspecting and cleaning** (tìm và xử lý các giao dịch bị thiếu thông tin), và **Moving data and building data pipelines** (xây dựng quy trình tự động để các kiểu dữ liệu truyền qua đó trở thành dữ liệu sạch).

Modeling

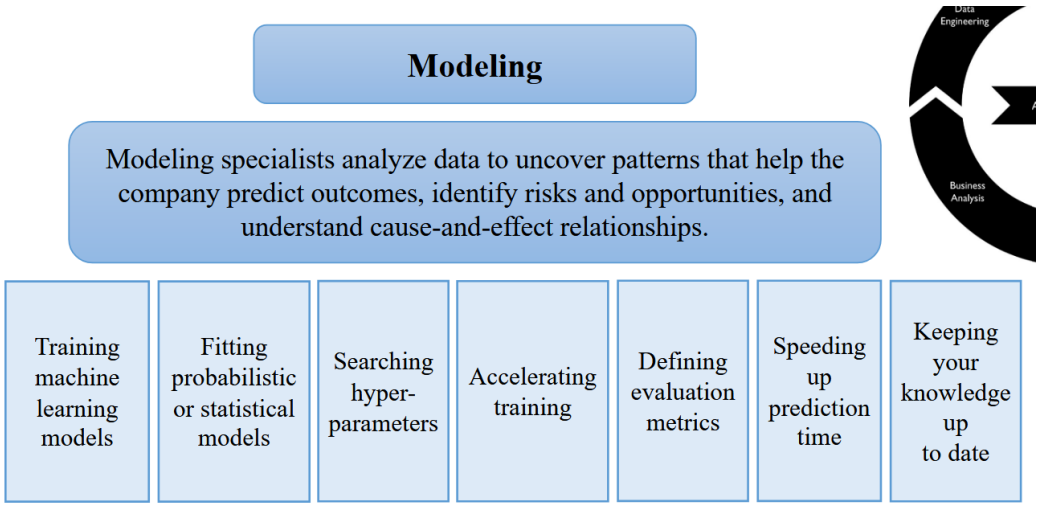


Figure 5: Modeling ML Model Pipeline

Đối với các Data Scientist họ sẽ đảm nhiệm **Training machine learning models** (dạy mô hình phân biệt giao dịch thật/giả), **Searching hyper-parameters** (tinh chỉnh mô hình để nó hoạt động tốt nhất), và **Defining evaluation metrics** (quyết định dùng thước đo nào để cân bằng giữa Bias và Varianece, ví dụ: "thà chặn nhầm còn hơn bỏ sót gian lận").

Deployment

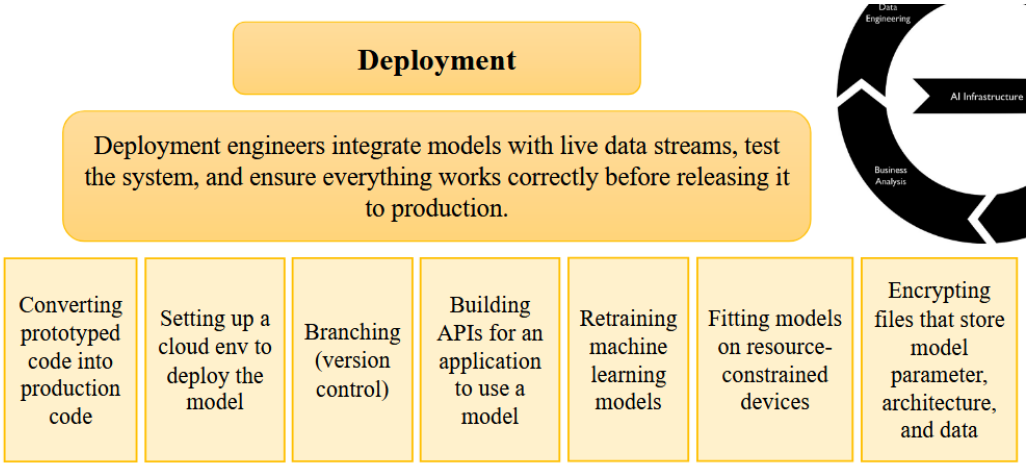


Figure 6: ML Model Deplopyment Pipeline

Để đưa mô hình vào thực tế, Kỹ sư AI sẽ: **Converting prototyped code into production code** (chuyển code Python thử nghiệm thành code tối ưu, chạy nhanh), **Building APIs** (xây dựng API để tích hợp mô hình vào ứng dụng ngân hàng), và **Setting up a cloud env** (thiết lập máy chủ trên Cloud để mô hình chạy 24/7).

Business analysis

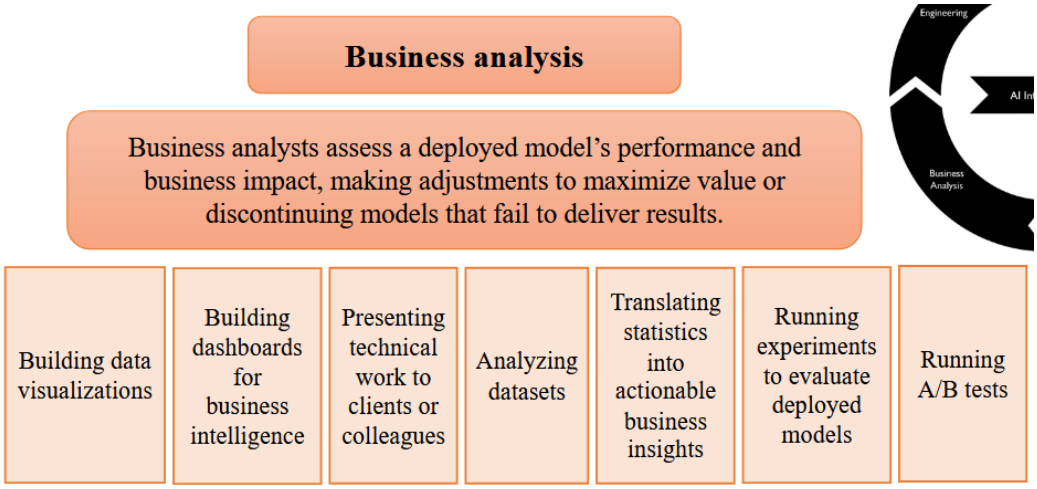


Figure 7: Business Analysis Pipeline

Đối với team phân tích nghiệp vụ, họ sẽ: **Building data visualizations** và **dashboards** (tạo biểu đồ theo dõi tỷ lệ phát hiện gian lận), **Running A/B tests** (so sánh mô hình mới và mô hình cũ xem cái nào hiệu quả hơn), và **Translating statistics into actionable business insights** (dịch các con số kỹ thuật sang ngôn ngữ kinh doanh, ví dụ: "Mô hình mới đã tiết kiệm được 10 tỷ VNĐ trong tháng qua").

AI Infrastructure - Cơ sở hạ tầng của 1 dự án AI

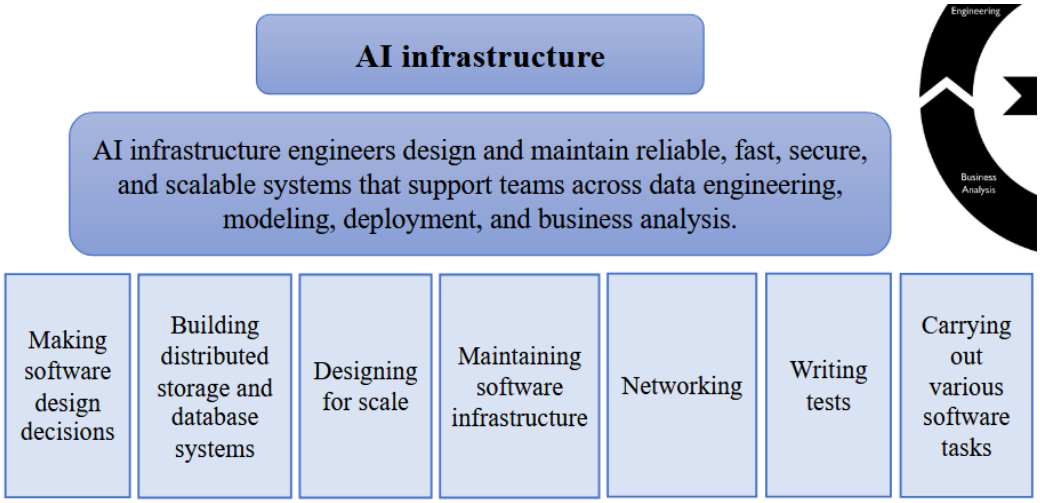


Figure 8: AI Infrastructure Pipeline

Đây là **nền tảng** cho tất cả các giai đoạn trên. Để hệ thống gian lận hoạt động, đội ngũ hạ tầng phải: **Building distributed storage** (xây dựng CSDL có thể lưu trữ hàng tỷ giao dịch), **Designing for scale** (thiết kế để hệ thống không bị sập khi có hàng triệu người giao dịch cùng lúc), và **Maintaining software infrastructure** (bảo trì máy chủ, mạng...).

6 vai trò trong nhóm phát triển AI

Dựa trên các giai đoạn của vòng đời dự án, chúng ta có thể thấy 6 vai trò chính trong một nhóm AI và sự tham gia của họ vào các giai đoạn khác nhau. Sơ đồ dưới đây minh họa sự phân bổ trách nhiệm của từng vai trò:

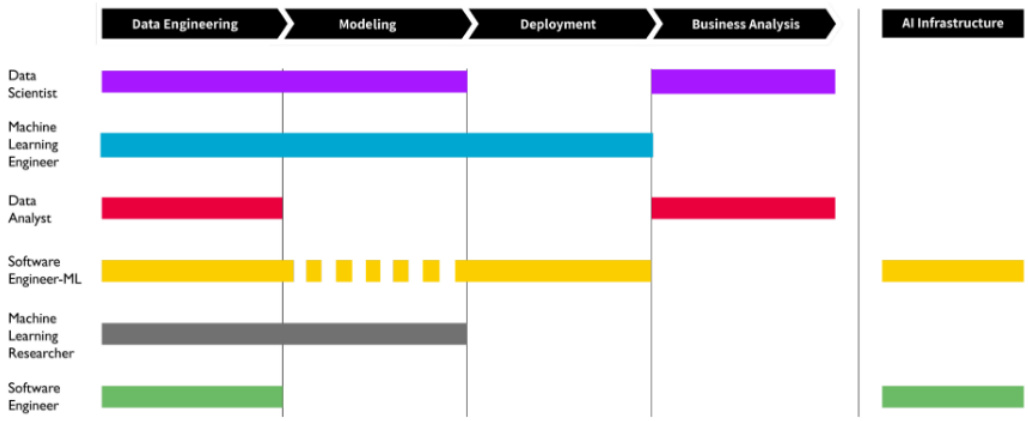


Figure 9: Sơ đồ phân bổ vai trò trong Vòng đời AI

- **Data Scientist (Nhà khoa học dữ liệu):** Tham gia sâu vào **Data Engineering** (chuẩn bị, khám phá dữ liệu) và **Modeling** (xây dựng mô hình). Họ cũng đóng vai trò quan trọng trong giai đoạn **Business Analysis** để đánh giá kết quả và tác động kinh doanh.
- **Machine Learning Engineer (Kỹ sư học máy):** Đây là vai trò trải dài nhất, tham gia vào *tất cả bốn giai đoạn* của vòng đời, từ **Data Engineering**, **Modeling**, **Deployment** cho đến **Business Analysis**. Họ là cầu nối giữa khoa học dữ liệu và kỹ thuật phần mềm.
- **Data Analyst (Nhà phân tích dữ liệu):** Tập trung chủ yếu vào hai giai đoạn đầu và cuối: **Data Engineering** (để hiểu và chuẩn bị dữ liệu cho việc phân tích) và **Business Analysis** (để phân tích, trực quan hóa và báo cáo kết quả kinh doanh).
- **Software Engineer-ML (Kỹ sư phần mềm - Học máy):** Tham gia vào **Data Engineering**, **Deployment** (xây dựng API, tối ưu hệ thống), và **AI Infrastructure**. Họ cũng có thể tham gia một phần vào **Modeling** (thể hiện bằng đường đứt nét), thường là để tối ưu hóa mô hình cho việc triển khai.
- **Machine Learning Researcher (Nhà nghiên cứu học máy):** Tập trung chuyên sâu vào hai giai đoạn đầu: **Data Engineering** và **Modeling**, nơi họ phát triển các thuật toán và kiến trúc mô hình mới.
- **Software Engineer (Kỹ sư phần mềm):** Vai trò này tập trung chủ yếu vào **AI Infrastructure** (xây dựng hệ thống nền tảng, CSDL) và có thể hỗ trợ một phần trong **Data Engineering**.

Phần 3: Feature Store và Feast là gì ?

3.1 Overview: Vấn đề cốt lõi của Vòng đời ML

Như đã đề cập ở phần Deployment, thách thức kỹ thuật lớn và phổ biến nhất trong các dự án ML là **Training-Serving Skew**.

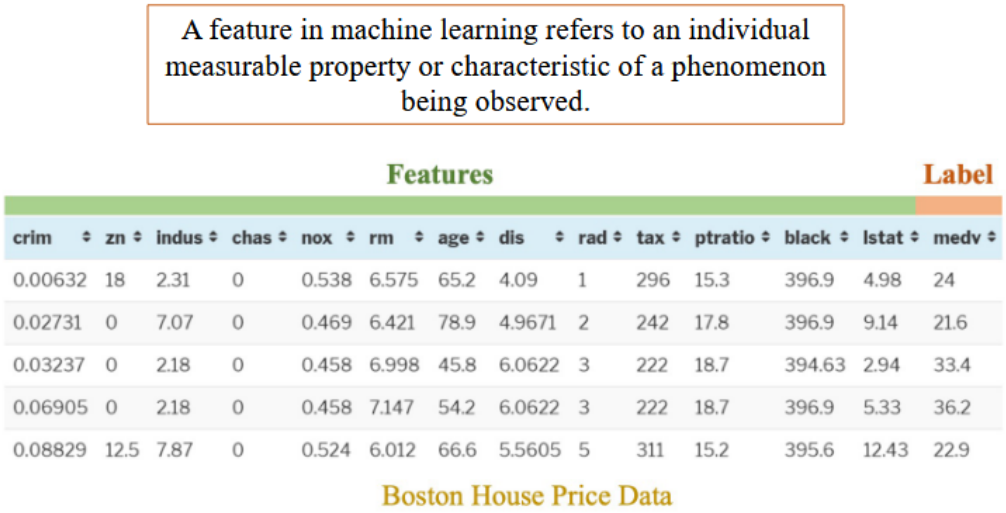
Đây là hiện tượng hiệu suất mô hình sụt giảm trên thực tế do sự khác biệt trong logic xử lý đặc trưng (feature) giữa hai môi trường:

- **Môi trường Training (Offline):** Các nhà khoa học dữ liệu xử lý dữ liệu theo lô (batch processing) để huấn luyện mô hình, thường dùng Python và Pandas.
- **Môi trường Serving (Online):** Các kỹ sư triển khai mô hình để dự đoán thời gian thực (real-time serving), đòi hỏi độ trễ cực thấp và có thể được viết bằng ngôn ngữ khác (Java, C++).

Sự thiếu nhất quán trong logic tính toán (ví dụ: cách xử lý giá trị null, làm tròn số) giữa hai hệ thống độc lập này là nguyên nhân chính gây ra "feature mismatch" (sai lệch đặc trưng), làm giảm hiệu suất của mô hình.

3.2 Feature Store: Giải pháp cho sự nhất quán

Trước tiên, một **feature (đặc trưng)** là một thuộc tính hoặc đặc điểm có thể đo lường được của một đối tượng quan sát, được dùng làm tín hiệu cho mô hình. Ví dụ, trong bộ dữ liệu Boston House Price, 'crim' (tỷ lệ tội phạm) hay 'rm' (số phòng) là các đặc trưng (Features) để dự đoán 'medv' (giá nhà, tức là Label).



crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9

Figure 10: Định nghĩa Feature và Label.

Feature Store ra đời để giải quyết hai vấn đề chính:

1. **Vấn đề Kỹ thuật (Training-Serving Skew):** Nó hoạt động như một nền tảng trung tâm, cung cấp một nguồn dữ liệu đặc trưng *duy nhất* cho cả hai môi trường. Logic tính toán đặc trưng được định nghĩa một lần và tái sử dụng, đảm bảo tính nhất quán tuyệt đối.
2. **Vấn đề Tổ chức (Hợp tác):** Thay vì mỗi nhóm tự xây dựng lại đặc trưng từ đầu (gây lãng phí tài nguyên), Feature Store cung cấp một "danh mục" (catalog) chung cho phép các nhóm khám phá, chia sẻ và tái sử dụng các đặc trưng đã có.

3.2.1 Các thành phần cốt lõi của Feature Store

Một Feature Store hoàn chỉnh bao gồm 5 thành phần chính:

- **Serving (Phục vụ):** Đây là lớp giao diện cung cấp dữ liệu cho mô hình.
  - **Offline Serving:** Cung cấp dữ liệu lịch sử cho *Model Training* (huấn luyện). Dữ liệu này được dùng để huấn luyện lại mô hình theo lịch (ví dụ: hàng ngày) hoặc khi phát hiện data drift.
  - **Online Serving:** Cung cấp dự đoán thời gian thực cho *Model Serving* (dự đoán), đảm bảo độ trễ thấp và khả năng mở rộng.

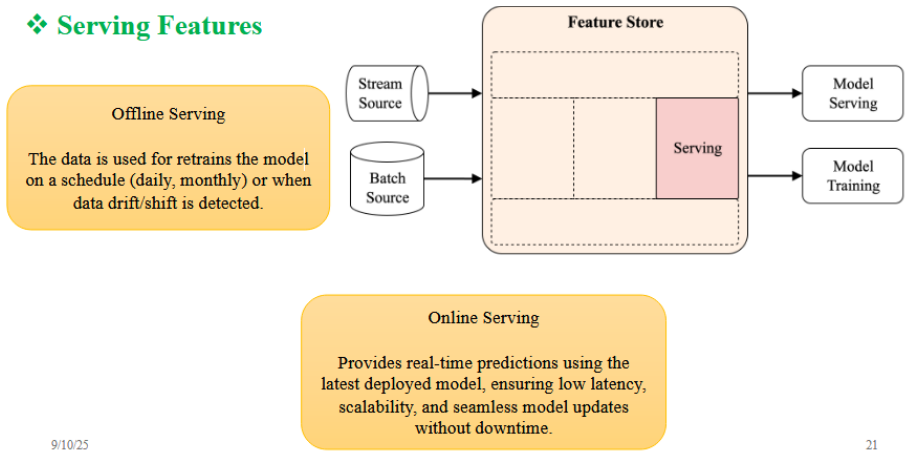


Figure 11: Hai chế độ Serving: Offline (cho Training) và Online (cho Serving).

- **Storage (Lưu trữ):** Lớp này trừu tượng hóa việc lưu trữ, chia làm hai kho:
  - **Offline Store:** Lưu trữ toàn bộ dữ liệu lịch sử của đặc trưng (có thể là hàng tháng, hàng năm) để huấn luyện. Thường là Data Warehouse (BigQuery, Snowflake) hoặc Data Lake (S3).
  - **Online Store:** Chỉ lưu trữ giá trị *mới nhất* của đặc trưng. Được tối ưu cho việc truy vấn tốc độ cao, độ trễ thấp (ví dụ: Redis, DynamoDB).

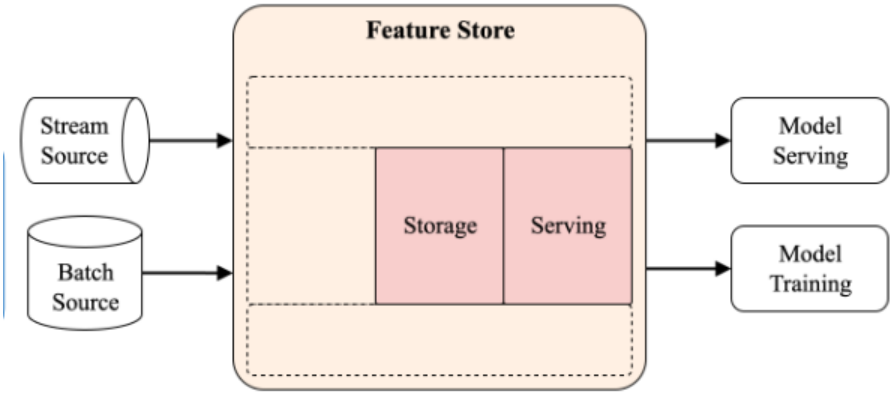


Figure 12: Hai thành phần lưu trữ: Offline Store và Online Store.

- **Transformation (Biến đổi):** Quản lý và điều phối các tác vụ tính toán đặc trưng từ dữ liệu thô.

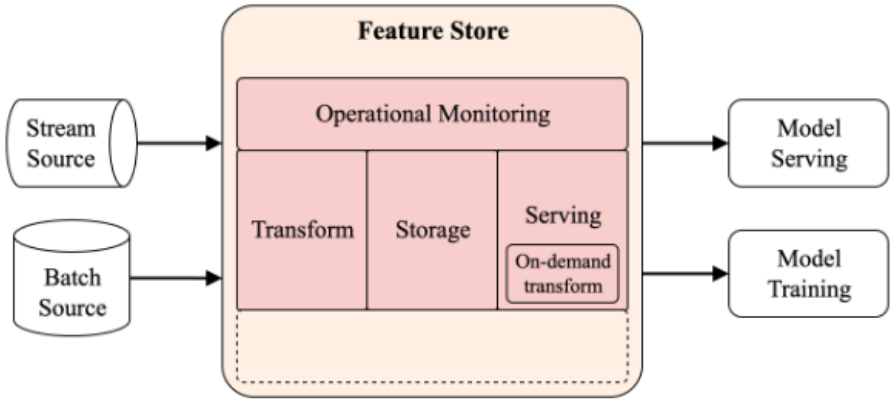


Figure 13: Lớp Transformation, Storage, Serving và Monitoring.

- **Monitoring (Giám sát):** Theo dõi sức khỏe của hệ thống. Quan trọng nhất là giám sát "data drift" (sự thay đổi trong phân phối thống kê của dữ liệu) và phát hiện "training-serving skew".

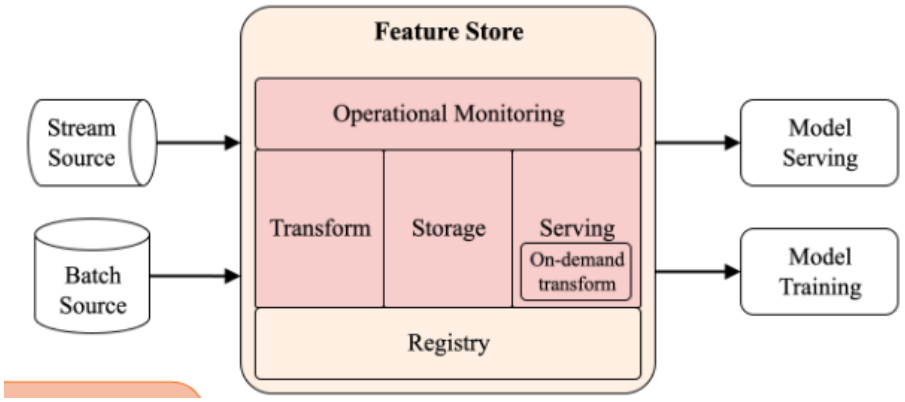


Figure 14: Lớp Monitoring giám sát chất lượng dữ liệu và vận hành.

- **Registry (Sổ đăng ký):** Là bộ não của Feature Store. Đây là thành phần trung tâm, đóng vai trò là "nguồn chân lý duy nhất" (single source of truth) lưu trữ mọi định nghĩa, siêu dữ liệu (metadata) về các đặc trưng.

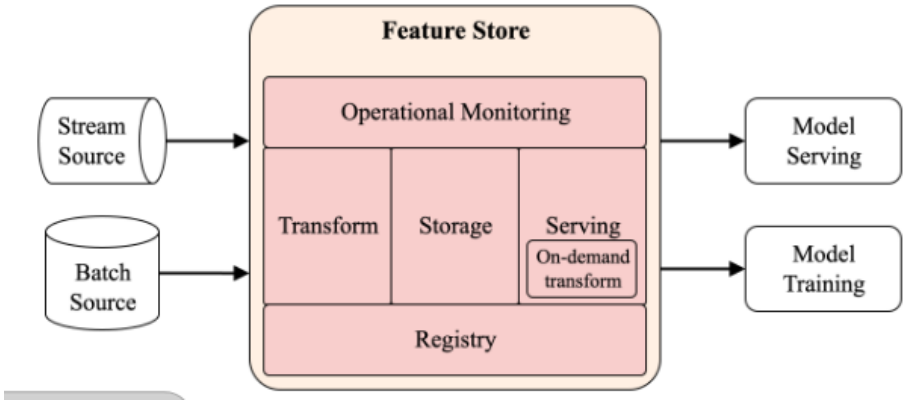


Figure 15: Registry là thành phần trung tâm lưu trữ mọi định nghĩa.

3.2.2 Lợi ích của Feature Store

Trong bối cảnh bài toán gian lận, việc áp dụng Feature Store mang lại các lợi ích rõ rệt:

- **Đảm bảo tính nhất quán:** Giải quyết triệt để Training-Serving Skew bằng cách đảm bảo tính nhất quán giữa dữ liệu huấn luyện (training) và dữ liệu dự đoán (inference).
- **Tăng tốc độ phát triển:** Cho phép theo dõi và chia sẻ đặc trưng giữa các nhà khoa học dữ liệu thông qua một kho lưu trữ có phiên bản.
- **Phục vụ đa mục đích:** Cung cấp đặc trưng cho nhiều trường hợp sử dụng ML: huấn luyện mô hình, xử lý theo lô, và dự đoán thời gian thực.
- **Giám sát chất lượng dữ liệu:** Giám sát chất lượng dữ liệu để nhanh chóng xác định các lỗi pipeline hoặc "data drift".

3.3 Feast là gì?

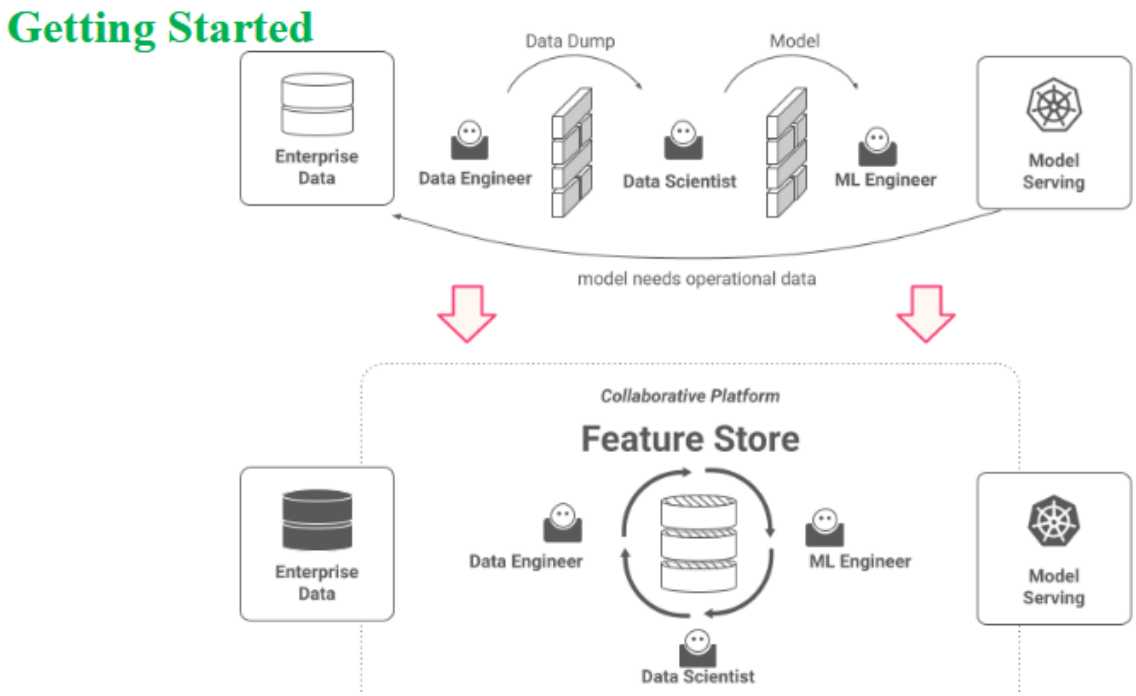


Figure 16: Triết lý của Feature Store: Chuyển từ quy trình riêng rẽ (trên) sang nền tảng hợp tác (dưới).

Như trong hình, quy trình truyền thống (ở trên) thường bị "vỡ": Kỹ sư dữ liệu (Data Engineer) thực hiện "Data Dump", trong khi Nhà khoa học dữ liệu (Data Scientist) và Kỹ sư ML (ML Engineer) làm việc riêng rẽ trên dữ liệu của họ. Tạo ra 1 bức tường ngăn cách phối hợp giữa 2 nhóm.

Feature Store (ở dưới) phá vỡ các bức tường này, tạo ra một **Nền tảng Hợp tác (Collaborative Platform)**, nơi tất cả các vai trò cùng tương tác với một kho lưu trữ đặc trưng duy nhất.

Feast là một Feature Store. Triết lý cốt lõi của Feast là nó **không phải là một nền tảng "tất cả trong một"**. Thay vào đó, nó được thiết kế như một **lớp điều phối mỏng và linh hoạt**, với mục đích kết nối và tái sử dụng hạ tầng dữ liệu mà tổ chức của bạn *đã có sẵn*.

- Feast **không** thay thế Data Warehouse (như Snowflake, BigQuery) → nó **sử dụng** chúng làm *Offline Store*.
- Feast **không** thay thế CSDL Key-Value (như Redis) → nó **sử dụng** chúng làm *Online Store*.

3.3.1 Kiến trúc và các thành phần của Feast

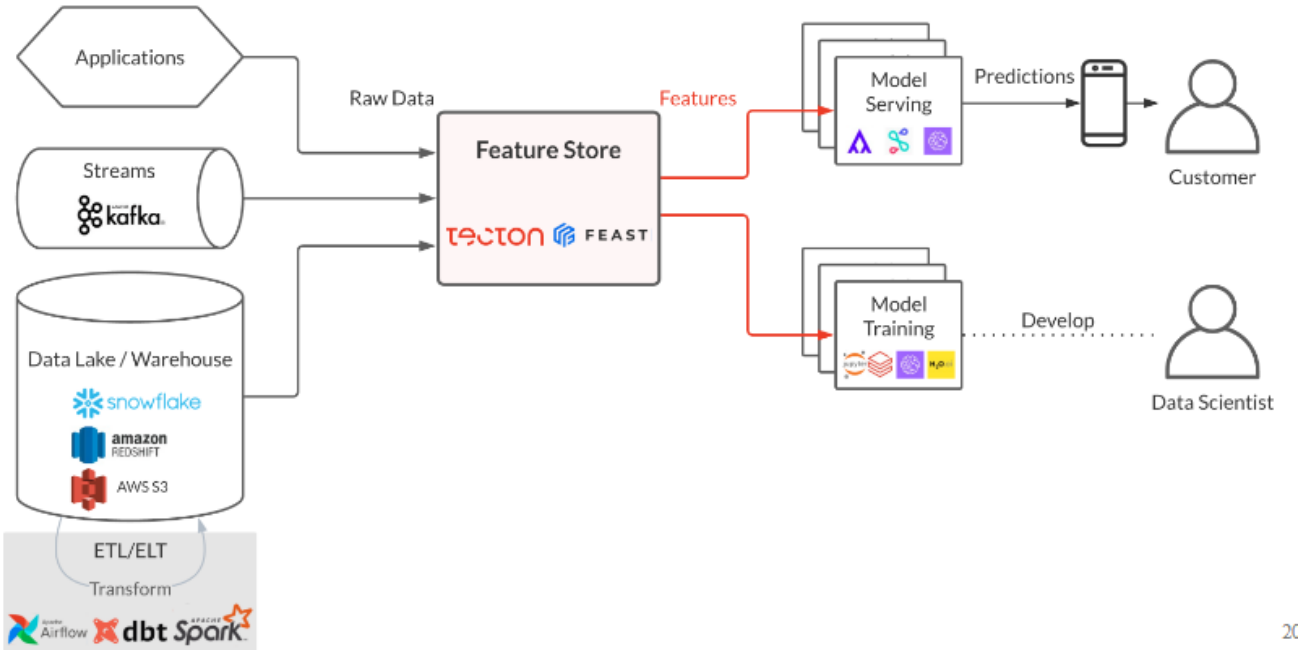


Figure 17: Kiến trúc tổng quan của Feast trong MLOps.

Kiến trúc của Feast thể hiện rõ triết lý của nó. Feast đóng vai trò trung tâm, tiếp nhận dữ liệu thô (Raw Data) từ nhiều nguồn đầu vào:

- **Data Lake / Warehouse** (ví dụ: Snowflake, Amazon Redshift, S3).
- **Streams** (ví dụ: Kafka).
- **Applications** (các ứng dụng nghiệp vụ).

Sau đó, Feast quản lý và phục vụ các Đặc trưng (Features) này cho hai đích đến:

1. **Model Training (Huấn luyện):** Cung cấp dữ liệu lịch sử (offline) cho các Nhà khoa học dữ liệu (Data Scientist) để phát triển (Develop) mô hình.
2. **Model Serving (Phục vụ):** Cung cấp dữ liệu (online) cho các mô hình đã triển khai để đưa ra dự đoán (Predictions) cho Khách hàng (Customer).

Để thực hiện điều này, Feast sử dụng các thành phần tương tự như một Feature Store lý thuyết, nhưng với các định nghĩa cụ thể:

- **Storage (Offline/Online Store):** Như đã nói, Feast *sử dụng* hạ tầng có sẵn của bạn. Trong file cấu hình, bạn sẽ chỉ định ‘offline_store‘ (ví dụ: một file Parquet, một bảng BigQuery) và ‘online_store‘ (ví dụ: Redis, SQLite).

- **Serving (Offline/Online Serving):** Feast cung cấp hai hàm SDK chính để truy vấn dữ liệu:
  - 'store.get_historical_features(...)': Dùng để truy vấn **Offline Store**. Hàm này thực hiện các phép join *point-in-time correct* (chính xác tại thời điểm) để chống rò rỉ dữ liệu khi huấn luyện.
  - 'store.get_online_features(...)': Dùng để truy vấn **Online Store**. Hàm này lấy vector đặc trưng mới nhất với độ trễ thấp nhất cho việc dự đoán thời gian thực.
- **Registry (Sổ đăng ký):** Đây là trái tim của Feast. Thay vì là một khái niệm trừu tượng, trong Feast, đây là một **feature repository** (kho chứa đặc trưng) mà bạn định nghĩa bằng code Python. Bạn sẽ khai báo các đối tượng như:
  - **DataSource:** Nơi dữ liệu của bạn được lưu trữ (ví dụ: file Parquet).
  - **Entity:** Đối tượng kinh doanh mà bạn muốn mô tả (ví dụ: 'customer_id' hoặc 'transaction_id').
  - **FeatureView:** Một nhóm logic các đặc trưng được liên kết với một Entity và một DataSource.

  Khi bạn chạy lệnh 'feast apply', các định nghĩa Python này sẽ được đăng ký vào file 'registry.db'.

Phần 4: Case Study

Phần 5: Final Revision: ôn lại mọi khái niệm trong bài cùng team Time Series